EMI REGISTRY MANUAL

EMIR Product Team

Document Version:	1.2.2
Component Version:	1.2.2
Date:	17 06 2013

This work is co-funded by the EC EMI project under the FP7 Collaborative Projects Grant Agreement Nr. INFSO-RI-261611.

Contents

1 Overview

1.1 EMIR Server (DSR or GSR)

EMI Service Registry is a Service Endpoint Registry designed and implemented during the EMI project. Its main goal is to discover all the Service Endpoints that exist. It consists of a collection of services that enables storing service records in a federated manner. Each of the record is a Service Endpoint Record (SER) complying with the OGF's GLUE 2.0 standard. The deployment of EMIR (which implies building an EMIR network over WAN) is bipartite: 1) Building a rooted hierarchy with a single EMIR server aggregating all the information within a federation 2) Sharing the information at the root level among peered EMIR servers (using P2P), thus enabling intra-federation discovery.

Feature Highlights:

- the service endpoint record registration includes the management of the services' endpoint information.
- Powerful data back-end based on MongoDB
- Schema-free information model based on JSON (using GLUE2 entity names for specific attributes)
- REST-ful API to browse the service registrations
- Security
  - PKI governed authentication
  - Policy based authorisation

For more information about EMIR, visit EMI's TWiki.

1.2 EMIR's Service Endpoint Record Publisher (EMIR-SERP)

The UMD services need to be registered into the EMI Registry service infrastructure to be discoverable for the clients. Most of the services or even the containers executing them provide a way to do this but not all of them. For those that are unable to register themselves automatically and periodically the EMIR-SERP is available.

The EMIR-SERP is a daemon like (background) service that can be executed next to these services (preferably on the same machine) and able to perform the automatic and periodical registration and update against the configured EMI Registry service on behalf of the service itself. This client uses exactly the same, standard RESTful API as the other clients do.

Most of the parameters of these registrations and updates can be configured. For the details see the Configuration section!

After the successful registration until the termination of the daemon, the EMIR-SERP client do the periodical updates then finally, when the execution of the daemon is over, it attempts to delete the service entries from the remote database.

The service entries can be defined in single files, in multiple files in a watchdir (that is periodically scanned for new files) or in resource BDIIs.

The entries can contain any kind of information allowed and accepted by the EMI Regisrty services and can be configured in the form of whole, formatted **json** documents or LDAP in case of BDII usage. The LDIF → JSON converson is also performed by EMIR-SERP.

2 Getting Started in 5 Minutes

2.1 Domain Service Registry (DSR)

This sections explains how to setup a Domain Service Registry (DSR) for a site. As a prerequisite, any SL6 host, either real of virtual is required.

2.1.1 Installing the DSR

Install the EMI release package

```
rpm -Uvh http://emisoft.web.cern.ch/emisoft/dist/EMI/2/sl6/x86_64/ ↩
    base/emi-release-2.0.0-1.sl6.noarch.rpm
```

Install the EMI Registry package.

```
yum install -y emi-emir
```

2.1.2 Configure the DSR

Edit the file

```
/etc/emi/emir/emir.config
```

and set the hostname and port.

```
emir.address=http://example.com:9126
```

Set the DSR parent attribute.

```
emir.parentAddress=http://parent.example.com:9126
```

Start the services

```
service mongod start
services emi-emir start
```

2.1.3 Test the DSR

Check that the DSR is running

```
http://example.com:9126/ping
```

2.2 Global Service Registry (GSR)

2.2.1 Installing the DSR

Same as DSR

2.2.2 Configuration

Edit the file

```
/etc/emi/emir/emir.config
```

and set the hostname and port.

```
emir.address=http://example.com:9126
```

set the global GSR flag

```
emir.global.enable=true
```

Start the services

```
service mongod start
services emi-emir start
```

2.2.3 Test the GSR

Check that the GSR is running

```
http://example.com:9126/ping
```

2.3 Service Endpoint Record Publisher (EMIR-SERP) with site BDII information source

2.3.1 Installing the EMIR-SERP

Install the emir-serp.

```
yum install emir-serp
```

Install the service translator

```
rpm -Uvh http://cern.ch/lfield/ginfo-0.1.5-1.noarch.rpm
```

2.3.2 Configure the EMIR-SERP (publisher)

Edit the file /etc/emi/emir-serp/emir-serp.ini and set the url for yor DSR.

```
url = http://example.com:9126
```

Set the json_dir_location

```
json_dir_location = /var/cache/emir-serp/services
```

Create the json directory.

```
mkdir -p /var/cache/emir-serp/services
```

Create a hourly cron job to run the following command, where *bdii.example.com* is the host name of a site BDII.

```
ginfo --host bdii.example.com --emi > /var/cache/emir-serp/services ↩
    /example
```

Start the service

```
service emir-serp start
```

2.3.3 Test the EMIR-SERP

Check the expected services are published

```
http://example.com:9126/services
```

2.4 Service Endpoint Record Publisher (EMIR-SERP) with resource BDII information source

2.4.1 Installing the EMIR-SERP

Install the emir-serp.

```
yum install emir-serp
```

2.4.2 Configure the EMIR-SERP (publisher)

Edit the file /etc/emi/emir-serp/emir-serp.ini and set the url for yor DSR and setup your credentials if needed.

```
url = http://example.com:9126
```

Set the resource_bdii_url variable.

```
resource_bdii_url = ldap://your.resource.bdii:2135/o=glue
```

Start the service

```
service emir-serp start
```

2.4.3 Test the EMIR-SERP

Check the expected services are published

```
http://example.com:9126/services
```

3 Installation

3.1 EMIR Server (DSR or GSR)

In order to install EMIR Server, it is a pre-requisite to install SUN or OpenJDK Java 6 (JRE or SDK). If not installed on the target system, it can be downloaded from http://java.oracle.com

- Linux based operating system
- MongoDB

EMIR is distributed in the following formats:

- Platform independent format, provided in "tar.gz" format
- RPM package, suitable SL5/SL6 and other Fedora based Linux derivatives (RedHat, CentOS etc. . .)
- Debian package

IMPORTANT NOTE ON PATHS
The location of the installation and configuration files differ depending on the type of bundle (see the above section).
If RPM bundle is being installed, the following paths will be used:

```
CONF=/etc/emi/emir
BIN=/usr/sbin
LOG=/var/log/emi/emir
LIB=/usr/share/emi/emir/lib
```

The platform independent binary places all the files under single directory. The contents will be:

```
CONF=INST/conf/
BIN=INST/bin/
LOG=INST/logs/
LIB=INST/lib/
```

The above variables (CONF, BIN, LOG, and LIB) will be used throughout the rest of this manual.

3.1.1 Installation using the RPM bundle (RedHat Distributions)

Download EMIR Server's RPM distribution from the EMI's emisoft and install it using the rpm or yum command.

Example

```
yum install -y emi-emir
```

3.1.2 Installation on Debian (Centos/Debian Distributions)

Download EMIR DEB distribution from the EMI's emisoft and install it using the apt-get command.

3.1.3 Database Installation

EMIR server uses MongoDB database as a backbone to store and indexe SER collections. The database dependency will automatically be fetched from the **emisoft** repository, while installing the EMIR Server. Otherwise it should be installed and configured before installing the EMIR. The installation and configuration instructions to setup the MongoDB database can be found on MongoDB's Web site.

3.1.4 Installation from the self-contained archive (tar.gz)

In order to generate, build and install the self contained binary it is required to follow the steps written below:

CREATING THE BUNDLE

1. check out the source code from **git://github.com/eu-emi/emiregistry.git**
2. go to **SOURCE_ROOT/emir-dist** directory
3. run **mvn assembly:assembly -DskipTests**

The archive can be found inside the **SOURCE_ROOT/target/emir-distribution-x.y.z-a-all.(tar.gz/zip)**, that contains all the necessary files for installation thus no special actions will be required except extraction to the target folder.

3.2 Installing the Publisher Client: EMIR-SERP

The installation of the EMIR-SERP client is trivial. The only thing to do is to install the `emir-serp` package from the EMI repository by executing:

```
yum install emir-serp
```

The package installation will provide the packages that are defined as dependencies, like python, python-ldap and python-simplejson if they are not previously installad on the machine.

4 EMIR Server Configuration

The EMIR server comes with a well documented configuration file (CONF/emir.config), containing a number of options to setup registry hierarchy, p2p, security, http server, and database. The settings in the configuration file are pre-defined to start-up the server in a non-production environment, however the administrator needs to review before deploying on the production (distributed) Grid environments.

4.1 General Configuration

The server configuration options in the CONF/emir.config are:

- Server address (plain or SSL)
- Settings of the type of the registry node, i.e. whether the current EMIR server instance is a child of some other (a parent) EMIR server node or a top/global registry in a hierarchy.

Property name	Type	Default value	Description
Server general settings			
`emir.address`	string	–	The address/URL of the EMIR server on which it receives registration and query requests. It should either start with http or https (SSL/TLS) mode, if "https" mode is selected the Authentication and Authorisation properties must be properly configured
`emir.anonymousPort`	Unsigned Integer	–	The anonymous http port number. Setting the property will start an additional *http* server (without SSL/TLS) only if the above server address is *https* (with SSL/TLS). It will provide *anonymous* access to the query interface (i.e. /services REST Web Service).

4.2 PKI Trust Settings Configuration

EMIR endorses Public Key Infrastructure (PKI) trust settings to validate certificates using EMI's caNL (JAVA version). The validation is performed when a connection with a remote peer is initiated over the network, using the SSL (or TLS) protocol, i.e. `emir.address` value has *https* scheme.

Certificates validation is primarily configured using a set of initially trusted certificates of so called Certificate Authorities (CAs). Those trusted certificates are also known as *trust anchors* and their collection is called as a *truststore*.

The validation mechanism except the *trust anchors* can use additional input for checking if a certificate being checked was not revoked and if its subject is in a permitted namesapce.

EMIR allows different types of truststores. All of them are configured using a set of specific properties in *CONF/emir.config* file.

4.2.1 OpenSSL Truststore

It allows using a directory with CA certificates stored in PEM format, with precisely defined names: Certificate Authorities (CA), Certificate Revocation List (CRL), signing policy and namespaces files are named as <hash>.0, <hash>.r0, <hash>.signing_policy and <hash>.namespaces respectively. Hash is the old hash of the trusted CA certificate subject name - in OpenSSL version newer than 1.0.0 use -suject_hash_old switch to generate it. If multiple certificates have the same hash then the default zero number must be incremented. It is suggested when a common truststore with EMI (and Globus) middlewares is needed.

4.2.2 Directory Truststore

It allows to use a list of wildcard expressions, concrete paths of files, or URLs to remote files as a set of trusted CAs and CRLs. The truststore is configured as a directory containing all the trusted certificates (or with a specified extension). The directory with stored IGTF trust anchors can be set as a EMIR truststore for instance.

4.2.3 Java Keystore (JKS) Truststore

A single repository (or a binary file) of X.509 public key certificates with (optionally) accompanying private key certificates. The Java JDK already bundles keytool utility - a certificate manage utility to create JKS truststores.

4.2.4 PKCS#12 Truststore

Similar to JKS trustore, single binary file can be used to store X.509 public with (optionally) accompanying private key certificates. The **OpenSSL pkcs12** command can be used to parse, read, and create these files; the extension for PKCS#12 files is ".p12".

Property name	**Type**	**Default value / mandatory**	**Description**
`emir.security.truststore.allowProxy`	[ALLOW, DENY]	`ALLOW`	Controls whether proxy certificates are supported.
`emir.security.truststore.type`	[keystore, openssl, directory]	*mandatory to be set*	The truststore type.
`emir.security.truststore.updateInterval`	integer number	`600`	How often the truststore should be reloaded, in seconds. Set to negative value to disable refreshing at runtime. *(runtime updateable)*

--- Directory type settings ---

Property name	Type	Default value / mandatory	Description
`emir.security.trustDirectoryConnectionTimeout`	integer number	`15`	Connection timeout for fetching the remote CA certificates in seconds.
`emir.security.trustDirectoryDiskCacheDir`	filesystem path		Directory where CA certificates should be cached, after downloading them from a remote source. Can be left undefined if no disk cache should be used. Note that directory should be secured, i.e. normal users should not be allowed to write to it.
`emir.security.trustDirectoryEncoding`	[PEM, DER]	`PEM`	For directory truststore controls whether certificates are encoded in PEM or DER.
`emir.security.trustStore.directoryLocation.*`	list of properties with a common prefix		List of CA certificates locations. Can contain URLs, local files and wildcard expressions. *(runtime updateable)*
--- Keystore type settings ---			
`emir.security.trustStore.keystoreFormat`	string		The keystore type (jks, pkcs12) in case of truststore of keystore type.
`emir.security.trustStore.keystorePassword`	string		The password of the keystore type truststore.
`emir.security.trustStore.keystorePath`	string		The keystore path in case of truststore of keystore type.
--- Openssl type settings ---			

Property name	Type	Default value / mandatory	Description
`emir.security.trust.crl.policy`	[GLOBUS_OPENSSL, EUGRIDPMA_GLOBUS, EUGRIDPMA, GLOBUS_EUGRIDPMA_REQUIRE, EUGRIDPMA_GLOBUS_REQUIRE, GLOBUS_REQUIRE, EUGRIDPMA_REQUIRE, EUGRIDPMA_AND_GLOBUS, EUGRIDPMA_AND_GLOBUS_REQUIRE, IGNORE]	EUGRIDPMA_GLOBUS	In case of openssl truststore, controls which (and in which order) namespace checking rules should be applied. The *REQUIRE* settings will cause that all configured namespace definitions files must be present for each trusted CA certificate (otherwise checking will fail). The *AND* settings will cause to check both existing namespace files. Otherwise OR is checked (in the order defined by the property).
`emir.security.trust.crl.openssl.path`	filesystem path	`/etc/grid-security/certificates`	Directory to be used for opeenssl truststore.
--- Revocation settings ---			
`emir.security.trust.crl.connectionTimeout`	integer number	15	Connection timeout for fetching the remote CRLs in seconds (not used for Openssl truststores).
`emir.security.trust.crl.diskCachePath`	filesystem path	-	Directory where CRLs should be cached, after downloading them from remote source. Can be left undefined if no disk cache should be used. Note that directory should be secured, i.e. normal users should not be allowed to write to it. Not used for Openssl truststores.
`emir.security.trust.crl.locations.*`	list of properties with a common prefix	-	List of CRLs locations. Can contain URLs, local files and wildcard expressions. Not used for Openssl truststores. *(runtime updateable)*

Property name	Type	Default value / mandatory	Description
emir.security.truststore.crlMode	[REQUIRE, IF_VALID, IGNORE]	IF_VALID	General CRL handling mode. The IF_VALID setting turns on CRL checking only in case the CRL is present.
emir.security.truststore.crlUpdateInterval	integer number	600	How often CRLs should be updated, in seconds. Set to negative value to disable refreshing at runtime. *(runtime updateable)*
emir.security.truststore.ocspCacheTtl	integer number	3600	For how long the OCSP responses should be locally cached in seconds (this is a maximum value, responses won't be cached after expiration)
emir.security.truststore.ocspDiskCache	filesystem path	-	If this property is defined then OCSP responses will be cached on disk in the defined folder.
emir.security.truststore.ocspLocalResponder.*	list of properties with a common prefix	-	Optional list of local OCSP responders
emir.security.truststore.ocspMode	[REQUIRE, IF_AVAILABLE, IGNORE]	IF_AVAILABLE	General OCSP ckecking mode. REQUIRE should not be used unless it is guaranteed that for all certificates an OCSP responder is defined.
emir.security.truststore.ocspTimeout	integer number	10000	Timeout for OCSP connections in miliseconds.
emir.security.truststore.revocation.order	[CRL, OCSP, OCSP_CRL]	OCSP_CRL	Controls overal revocation sources order
emir.security.truststore.revocation.useAll	[true, false]	false	Controls whether all defined revocation sources should be always checked, even if the first one already confirmed that a checked certificate is not revoked.

4.2.5 Examples

Directory truststore, with a minimal set of options:

```
emir.security.truststore.type=directory
emir.security.truststore.directoryLocations.1=/trust/dir/*. ↩
    pem
emir.security.truststore.directoryLocations.2=/other/dir/*. ↩
    pem
emir.security.truststore.crlLocations=/trust/dir/*.crl
```

Directory truststore, with complete set of options:

```
emir.security.truststore.type=directory
emir.security.truststore.allowProxy=DENY
emir.security.truststore.updateInterval=1234
emir.security.truststore.directoryLocations.1=/trust/dir/*. ↩
    pem
emir.security.truststore.directoryLocations.2=http:// ↩
    caserver/ca.pem
emir.security.truststore.directoryEncoding=PEM
emir.security.truststore.directoryConnectionTimeout=100
emir.security.truststore.directoryDiskCachePath=/tmp
emir.security.truststore.crlLocations=/trust/dir/*.crl http ↩
    ://caserver/crl.pem
emir.security.truststore.crlUpdateInterval=400
emir.security.truststore.crlMode=REQUIRE
emir.security.truststore.crlConnectionTimeout=200
emir.security.truststore.crlDiskCachePath=/tmp
```

Openssl truststore:

```
emir.security.truststore.type=openssl
emir.security.truststore.opensslPath=path/to/truststores/ ↩
    openssl
emir.security.truststore.opensslNsMode= ↩
    EUGRIDPMA_GLOBUS_REQUIRE
emir.security.truststore.allowProxy=ALLOW
emir.security.truststore.updateInterval=1234
emir.security.truststore.crlMode=IF_VALID
```

Java keystore used as a truststore:

```
emir.security.truststore.type=keystore
emir.security.truststore.keystorePath=path/to/truststores/ ↩
    emir-truststore.jks
emir.security.truststore.keystoreFormat=JKS
emir.security.truststore.keystorePassword=xxxxxx
```

4.3 Configuring the Credentials

EMIR uses private key and a corresponding certificate (called together as a *credential*) to identify clients and servers. The credentials can be provided in several formats. The following table list all possible variants and corresponding parameters.

Property name	Type	Default value / mandatory	Description
`emir.security.credential.path`	file system path	*mandatory to be set*	Credential location. In case of *jks*, *pkcs12* and *pem* store it is the only location required. In case when credential is provided in two files, it is the certificate file path.
`emir.security.credential.format`	[jks, pkcs12, der, pem]	–	Format of the credential. It is guessed when not given. Note that *pem* might be either a PEM keystore with certificates and keys (in PEM format) or a pair of PEM files (one with certificate and second with private key).
`emir.security.credential.password`	string	–	Password required to load the credential.
`emir.security.credential.keyPath`	string	–	Location of the private key if stored separately from the main credential (applicable for *pem* and *der* types only),
`emir.security.credential.keyPassword`	string	–	Private key password, which might be needed only for *jks* or *pkcs12*, if key is encrypted with different password then the main credential password.
`emir.security.credential.keyAlias`	string	–	Keystore alias of the key entry to be used. Can be ignored if the keystore contains only one key entry. Only applicable for *jks* and *pkcs12*.

4.3.1 Examples

Credential as a pair of DER files:

```
emir.security.credential.format=der
emir.security.credential.password=emi
emir.security.credential.path=path/to/credentials/cert-1.der
emir.security.credential.keyPath=path/to/credentials/pk-1. ↩
    der
```

Credential as a JKS file (type can be autodetected in almost every case):

```
emir.security.credential.path=path/to/credentials/server1. ↩
    jks
emir.security.credential.password=xxxxxx
```

4.4 ACL Based Authorization

The EMIR offers two alternative options to authorise its' clients.

- Using Access Control List (ACL)
- XACML Policy based authorization

This is the default mechanism to access control the *Create*,*Update*, and *Delete* operations on EMIR's SER database. The client SERP or child DSR registering SERPs with a parent DSR/GSR get authorised while matching it's distinguished name (DN) against the pre-defined ACL file (CONF/emir.acl). Whereas the file contains a list of DN and role pairs, separated by *::* symbol, see the example below:

the property in the `CONF/emir.config` file

Property name	Type	Default value	Description
`emir.security.access.filesystem.acl`	filesystem path	`CONF/emir.acl`	The location of the ACL file

Example ACL file contents

```
emailAddress=emiregistry@user.eu,CN=EMIRegistry-Demo-User,OU=JSC,O= ↩
    Forschungszentrum Juelich GmbH,L=Juelich,C=DE :: serviceowner
emailAddress=emiregistry@user.eu-admin,CN=EMIRegistry-Demo-User- ↩
    Admin,OU=JSC,O=Forschungszentrum Juelich GmbH,L=Juelich,C=DE :: ↩
     admin
```

The public key certificate or DN should be sent to the DSR administrator for successfull SER registrations.

Roles: There are only two pre-defined roles within the scope of ACL file:

- a user with the `admin` role is considered as a super user who can change any registration, owned by anyone,
- whereas the `serviceowner` is allowed to create or modify the (owned) existing SERPs.

Important
The ACL based authorisation is only (and automatically) activated when the DSR/GSR is running on SSL/TLS mode

4.5 Policy Based Authorization with XACML

Using XACML 2.0 is an alternative way to authorise clients (User, EMIR-SERP, DSR, or GSR) in a fine grained manner. The administrator should review the policies defined in the `CONF/xacml2Policies/` folder and change them according to her infrastructure needs. However the already defined policies provides a good starting point to the administrators to define/modify the policies.

Important
the XACML policy based authorisation will be ignored, if the ACL based authorisation is activated

In order to enable the XACML based authorization: attribute sources and policies must be configured.

4.5.1 Setting Attribute Sources

EMIR currently supports the file based attribute sources; the client DNs can be included in the attributes file.

Table 1: File Attribute Source Settings

Property name	Type	Default value	Description
`emir.security.attributes.order`	string	`FILE`	This property is a space separated list of attribute source names, which are then configured in detail below. The named attribute sources are queried in the given order.
`emir.security.attributes.FILE.class`	string	`eu.emi.emir.aip.FileAttributeSource`	configuration of the *FILE* attribute source
`emir.security.attributes.FILE.matching`	[strict, regexp]	`strict`	Specifies the matching or client DNs
`emir.security.attributes.FILE.conf`	filesystem path	`CONF/users/testDNs.xml` or `CONF/users/testDNs-regexp.xml`	The path to the file containing subjects' DNs. The file suffixed with `strict` use strict checking of DNs, whereas the file suffixed with `regexp` contain entries using regualr expressions

4.5.2 Setting XACML Policies

Table 2: Policy settings

Property name	Type	Default value	Description
`emir.security.accesscontrol.pdp.conf`	filesystem path	`CONF/xacml2.conf`	The path to the XACML2 configuration, containing the rules of executing the policies
`emir.security.accesscontrol.pdp`	string	`eu.unicore.uas.pdp.local.LocalHerasafPDP`	The name of the pdp class to endorse, for the xacml2 policies execution

The `CONF/xacml2.config` file contains raw xacml polices, enable EMIR (DSR or GSR) administrators to write their own rules.

4.6 MongoDB Database Configuration

The EMIR uses MongoDB to store and index the SER collections. It must be configured and running before deploying any EMIR (DSR or GSR) server.

Property name	Type	Default value	Description
Connection Settings			
`emir.mongodb.hostName`	string	`localhost`	Fully qualified host name of the machine on which MongoDB is setup
`emir.mongodb.port`	Integer	`27017`	The port number
Database Settings			
`emir.mongodb.dbName`	string	`emiregistry`	The name of the database to store the SERP records
`emir.mongodb.colName`	string	`services`	The name of the collection (of the database) in which the records will be stored
Login Settings			
`emir.mongodb.userName`	string	–	The username to access the MongoDB database
`emir.mongodb.password`	string	–	The password to access the MongoDB database

For high loads, especially at the GSR level, it is recommended to setup MongoDB replication for enhanced scalability and performance.

4.7 Building EMIR Network

EMIR allows building a network of registries participating in a Grid infrastructure or federation. The network can be of type hierarchical or Peer-to-Peer (P2P). In an hierarchical network, the SER collections are propagated from leaf DSR node to the top level root node, called GSR. Each DSR has only one parent, either DSR or GSR to which it pushes it's SER collections. At the root level the P2P network of GSR is formed to replicate the SER collections among multiple GSRs by referring a pre-configured `Global list`. The global list contains a listing of URLs of all the GSRs, each of which should be able to access the URLs.

Note
The machines running the EMIR servers should be time synchronised, either by NTP or any alternative mechanism

4.7.1 How to Setup DSR?

In order to build hierarchy of DSRs must be able to propagate the SER collections to any **single** parent DSR or a GSR.

Table 3: Parent DSR Settings

Property name	Type	Default value	Description
EMIR's DSR settings			
`emir.parentAddress`	string	–	The address/URL (http or https) of the EMIR DSR server to which it propagates its SER collection

Important
Add DN of child DSR into the parent DSRs CONF/emir.acl or CONF/users/testUd-(strict | regsexp).xml

4.7.2 How to Setup GSR?

The root level GSR has two primary functions:

- aggregation of children DSR SER collections
- replicating the SER collections among other GSRs (visible of *Global List*)

Table 4: GSR Settings

Property name	Type	Default value	Description
General GSR Settings			

Table 4: (continued)

Property name	Type	Default value	Description
`emir.global.enable`	boolean	`false`	If set to true, indicating the registry node is global. It will then replicate the state among peer global registries (GSRs), the *emir.parentAddress* property will be ignored (if enabled), as the root registry should not contain any parent.
`emir.global.sparsity`	Unsigned Integer	`2`	It determines the number of neighbors as a function of the actual number of member nodes of the network.
`emir.global.retry`	Unsigned Integer	`5`	It specifies a number of attempts if communication to another GSR is failed.
`emir.global.etValid`	Unsigned Integer	`12`	Specifies period in hours for checking the entries in the soft state database and strip the expired entries (but still keeps them).
`emir.global.softStateDelay`	Integer	`2`	Extend the expiration time with this time delay in hours.
`emir.global.etRemove`	Integer	`24`	Specifies period in hours for checking the entries in the soft state database and remove the expired entries.
Global List Settings			
`emir.global.providerURL`	URL or filesystem path	–	Link to the document listing GSR URLs. The URL(s) is/are important for building the GSR's P2P network at the global level.

Important
Add DN of child DSR into the GSRs CONF/emir.acl or CONF/users/testUd-(strict | regsexp).xml

4.8 Service Endpoint Record (SER) Management

4.8.1 Setting Service Endpoint Records (SER) Lifetime

In EMIR, every SER has associated lifetime or (Time-To-Live) TTL. The settings can be defined in DSR or GSR to restrict the maximum assignable lifetime and assign default lifetime if missing from the registration.

Table 5: SER TTL Settings

Property name	Type	Default value	Description
`emir.record.expiryMaximum`	Unsigned Integer (in days)	–	Maximum assignable lifetime for the SERs containing the Service_ExpireOn property, defined in days, minimum value: 1.
`emir.record.expiryDefault`	Unsigned Integer (in days)	–	The default lifetime will be set from the given property if the incoming registration is without the Service_ExpireOn attribute.

4.8.2 Filtering Service Endpoint Records (SER)

EMIR offers a way to block

- SERs from being registered via DSR or EMIR-SERP
- SERs from being propagated to it's parent DSR or GSR

Table 6: SER Filter Settings

Property name	Type	Default value	Description
`emir.record.blockListFilesystem`	filesystem path	`$GLITE_LOCATION/CONF/inputfilter.ini`	The file containing list of SER IDs, matching services will be blocked from registration to it's index
`emir.record.blockListFilesystem`	filesystem path	`$GLITE_LOCATION/CONF/outputfilter.ini`	The file containing list of SER IDs, matching services will be blocked from propagation to it's parent DSR

4.8.3 Validation of Mandatory Attributes

Usually the DSR or GSR does not allow the SER to be registered(or updated) without having mandatory attributes. This validity check can be disabled to allow the publishers to register a SER with custom attributes to the EMIR server. Hence providing a flexibility to the publishers, whereas the consumers have to examine all the attributes while performing some operation on the service (contained in the SER) itself.

Table 7: Enable/Disable Validity Checks

Property name	Type	Default value	Description
`emir.record.attributeChecking`	String	`strict`	There are two possible modes: *strict* or *flexible*. If set to *strict* the emir server will check mandatory attributes in the record being updated or registered. If set to *flexible* only SERVICE_ENDPOINT_ID will be taken as a mandatory attribute.

4.9 Logging Configuration

The EMIR server uses log4j to provide log facilities to record all but some of the server activities. In order to change the logging configuration, CONF/log4j.properties should be reviewed by the administrator.

4.10 Advanced HTTP Server Settings

EMIR uses Eclipse's Jetty server to host REST Web services. Following table lists the important properties.

Important
Do not set **emir.jetty.requireClientAuthn** and **emir.jetty.wantClientAuthn** in CONF/emir.config file, as they are automatically set by the EMIR server on start-up.

Property name	Type	Default value / mandatory	Description
`emir.jetty.disableCipherSuites`	string	*empty string*	Space separated list of SSL cipher suites to be disabled.
`emir.jetty.fastRandom`	[true, false]	`false`	Use insecure, but fast pseudo random generator to generate session ids instead of secure generator for SSL sockets.
`emir.jetty.gzip.enable`	[true, false]	`false`	Controls whether to enable compression of HTTP responses.
`emir.jetty.gzip.minGzipSize`	integer number	`100000`	Specifies the minimal size of message that should be compressed.
`emir.jetty.highLoadConnections`	integer >=1	`200`	If the number of connections exceeds this amount, then the connector is put into a special *low on resources* state. Existing connections will be closed faster. Note that this value is honored only for NIO connectors. Legacy connectors go into low resources mode when no more threads are available.

Property name	Type	Default value / mandatory	Description
emir.jetty.lowResourceMaxIdleTime	integer >= 1	100	In low resource conditions, time (in ms.) before an idle connection will time out.
emir.jetty.maxIdleTime	integer >= 1	200000	Time (in ms.) before an idle connection will time out. It should be large enough not to expire connections with slow clients, values below 30s are getting quite risky.
emir.jetty.maxThreads	integer >= 1	255	Maximum number of threads to have in the thread pool for processing HTTP connections.
emir.jetty.minThreads	integer >= 1	1	Minimum number of threads to have in the thread pool for processing HTTP connections.
emir.jetty.requireClientAuthn	[true, false]	true	Controls whether the SSL socket requires client-side authentication.
emir.jetty.soLingerTime	integer number	-1	Socket linger time.
emir.jetty.useNIO	[true, false]	false	Controls whether the NIO connector be used. NIO is best suited under high-load, when lots of connections exist that are idle for long periods.
emir.jetty.wantClientAuth	[true, false]	true	Controls whether the SSL socket accepts (but does not require) client-side authentication.

5 EMIR-SERP Configuration

The configuration of EMIR-SERP can be performed by editing its configuration file or files. The configuration can be found basically in one file that default location is `/etc/emi/emir-serp/emir-serp.ini`.

This file contains every configuration options that can be the EMIR-SERP daemon control by, like *service url*, *logging verbosity*, *credential location*, etc.

The advanced service entries to be propagated can be described in separated configuration files preferably also under this directory and use to have `.json` extension.

The main configuration file has INI format. The `emir-serp` section contains the daemon scoped options while the others are to describe the different service entries to be registered. In these cases the exact name is indifferent, they just have to differ from each other and must avoid the `emir-serp` name as well.

5.1 Configuration options

Note
The names of options are case-insensitive.

5.1.1 url

`Location:` `emir-serp` section

`Default value:` *No default value*

`Mandatory:` Yes

`Description:`

URL of the EMIR service to connect in a protocol://domain:port format.

If protocol is missing default `https` is used. If port is missing default `54321` is used. The domain part is mandatory.

Examples

```
url = emiregistry2.grid.niif.hu
url = https://emiregistry2.grid.niif.hu
url = https://emiregistry2.grid.niif.hu:54321
```

5.1.2 period

`Location:` `emir-serp` section

`Default value:` *No default value*

`Mandatory:` Yes

`Description:`

The period of the registration/update messages. Its value is given in hours.

5.1.3 validity

`Location:` `emir-serp` section

`Default value:` *No default value*

`Mandatory:` Yes

`Description:`

The validity of the registration entries. Its value is given in hours.

5.1.4 cert

`Location:` `emir-serp` section

`Default value:` /etc/grid-security/hostcert.pem

`Mandatory:` No

`Description:`

User certificate file location in PEM format. Only used and checked if the protocol in the url option is *https*.

5.1.5 key

`Location:` `emir-serp` section

`Default value:` /etc/grid-security/hostkey.pem

`Mandatory:` No

`Description:`

User key file location in PEM format. Only used and checked if the protocol in the url option is *https*.

5.1.6 cadir

`Location:` `emir-serp` section

`Default value:` /etc/grid-security/certificates

`Mandatory:` No

`Description:`

A path pointing to the store where the PEM certificate of the trusted Certificate Authorities can be found. Only used and checked if the protocol in the url option is *https*.

5.1.7 verbosity

`Location:` `emir-serp` section

`Default value:` error

`Mandatory:` No

`Description:`

Logging verbosity. The parameter is optional. If missing or an invalid value is given, the default value will be used. The logs are written into the log file that can be found in the */var/log/emi/emir-serp* directory by default.

Note

The service entries can be defined in separated ini sections. The name of the section is irrelevant but must be different in every cases!

Any of *json_file_location*, *json_dir_location* or *resource_bdii_url* must be present in a section to enable EMIR-SERP registration otherwise section is going to be skipped.

5.1.8 json_file_location

`Location:` service related section

`Default value:` *No default value*

`Mandatory:` Yes

`Description:`

The service entry can be defined in a single external json formatted file per service. Any allowed json attributes are allowed in this way. The location of this file must be defined in the *json_file_location* ini variable.

The value of `json_file_location` is used only if no `resource_bdii_url` or `json_dir_location` are present in the same section.

5.1.9 json_dir_location

`Location:` service related section

`Default value:` *No default value*

`Mandatory:` Yes

`Description:`

Multiple entries belonging to a service can be put into separated json files in a common directory. The script periodically scan the content of the directory setted up with this attribute and the content of the found json files will be propagated to the EMIR service.

The value of `json_dir_location` is used only if no `resource_bdii_url` is present.

5.1.10 resource_bdii_url

`Location:` service related section

`Default value:` *No default value*

`Mandatory:` Yes

`Description:`

The service information to be registered can be harvested from directly from resource BDII LDAP servers. EMIR-SERP periodically queries the remote database, converts the result, and publish the service information to the previously configured EMIR service.

If *resource_bdii_url* attribute is present both *json_dir_location* and *json_file_location* are ignored.

If port is missing default 2170 is used. If LDAP base is missing default *o=glue* is used. Only *ldap* scheme is accepted in the URL.

6 How to use EMIR API?

The EMI Registry allows Services to register/publish their capabilities while the Service Consumers are able to find the deployed services.

This section contains the description of the REST-ful interface, that allows the management of the service information (or entries) by exposing the individual URIs. The normative description of the API cab also be defined as Web Application Description Language (WADL) document WADL Section 8.

6.1 Creatung OR Updating the Service Endpoint Records

`HTTP Method` : PUT

`URI` : /serviceadmin

`Content Type` : application/json

`Security Implications` : Requires an authenticated "and" authorized client's access to perform this operation

6.1.1 Request

The request body contain a similar JSON array object as defined in Section 7, it contains description of the Services to be updated or created. The endpoint records will be updated automatically, if the JSON document in the request body is already existing. The operation takes place only after the successful authentication and authorisation checks.

6.1.2 Response

The response contains an array of JSON Objects as it was sent in the request, confirming the successful update.

`Status Code` : OK / 200

6.2 Delete existing Services

`HTTP Method` : DELETE

`URI` : /serviceadmin

`Security Implications` : Requires an authenticated "and" authorized user access to perform this operation

6.2.1 Request

The Service Entry matching the Endpoint ID will be deleted from the registry only if the client executing the action has authorised access and the method is allowed by the security plugins.

`URL Query Parameters` : Service_Endpoint_ID= <*Service Endpoint ID*>

Example : /serviceadmin?Service_Endpoint_ID=urn:endpoint:emi1

6.2.2 Response

`Status Code` : OK / 200

6.3 Query for Endpoint Information

`HTTP Method` : GET

`URI` : /services

`Content Type` : application/json

6.3.1 Request

The request contains the key-value pairs separated by ampersand `&`

`Query Parameters` : AttributeName=<Attribute_Value>&AttributeName=<Attribute_Value>&...

Example : /services?Service_Type=eu.emi.es&Service_Endpoint_HealthState=ok

The response contains an array of service entries packed in a JSON array object

`Status Code` : OK / 200

6.4 Rich Querying in EMIR

`HTTP Method` : GET

`URI` : /services

`Content Type` : application/json

6.4.1 Request

The request contains the JSON document including with support for defining advanced clauses, the http://www.mongodb.org/display/DOCS/Advanced+Queries, MongoDB Advanced Queries[MongoDB JSON Query Language] describes the various types of queries

Additional keys (skip, limit) can also be added to paginate the returning results.

6.4.2 Response

The response contains the array of service entries packed in a JSON array object

`Status Code` : OK / 200

6.5 Querying the EMIR for GLUE 2.0 XML Documents

`HTTP Method` : GET

`URI` : /services

`Content Type` : application/xml

6.5.1 Request

The request contains the key-value pairs separated by ampersand `&`

`Query Parameters` : AttributeName=<Attribute_Value>&AttributeName=<Attribute_Value>&...

Example : /services?Service_Type=eu.emi.es&Service_Endpoint_HealthState=ok

6.5.2 Response

The response contains an XML document containing service entries in GLUE 2.0 format

`Status Code` : OK / 200

6.6 Rich Querying the EMIR for GLUE 2.0 XML Documents

The request and response interface is same as defined above, however the content type must be defined as **application/xml** instead.

6.7 Traversing through Query Results

It is very likely that a client (plain or rich) query can evaluate to a huge number of result set, which may result in memory over-flow and delayed response. In order to make the listing scalable and faster response times of the query requests, a traversal mechanism has been implemented at the Web services as well as JAVA client layer. It allows a client to provide specific query parameters, while offering the two alternative ways, thus highly depending on the size of EMIR index.

Note
The same iterating parameters can be used for the available query methods, such as, with query parameters or rich querying

---

6.7.1 Using `skip` and `limit` Query Parameters

skip defines an offset or a number of records to be skipped from the query result

limit defines a total number of resulting endpoints expected from the query

Following example illustrates this method of traversal:

Request

`GET / POST`: /services/?skip=<Non-Negative Number>&limit=<Non-Negative Number>

Example Usage: https://emir.example.org/services?skip=10&limit=100

`Content-Type`: application/json OR application/xml

Response

An array of matching endpoint records, either in a JSON or an XML format depending on the Content-Type header

This method is suitable when size of endpoint records stored in the EMIR server is not larger than 1000.

6.7.2 Using `pageSize` and `ref` Query Parameters

Being a memory efficient and robust, this method is ought to be the most preferable way of traversing the endpoint records.

pageSize defines a total number of resulting endpoint records expected from a query

ref every returned page contains a reference or pointer to the next page for further traversal

Following example illustrates this method of traversal:

Request

`GET / POST:` /services/pageSize=<Non-Negative Number>&ref=<String>

Example Usage: https://emir.example.org/services?pageSize=10&ref=87701693-ca33-482c-bbf4-843f9952e012

`Content-Type:` application/json OR application/xml

Response

An array of matching endpoint records, either in a JSON or an XML format depending on the Content-Type header

6.8 Viewing the Service information template

This To view the GLUE 2.0's JSON flavored service model.

`HTTP Method` : GET

`URI` : /model

`Content Type` : application/json

6.8.1 Request

N/A

6.8.2 Response

A JSON document containing all the mandatory and optional attribute

6.9 Information about the Deployed EMIR Server

There is an Web services interface to query the EMIR server's status in JSON format. This specifically contains the following attributes:

- EMIRServerVersion
- MongoDBVersion
- JavaVersion
- OSName
- OSArchitecture
- OSVersion
- EMIRServerComponentName
- AnonymousAccessPortNumber
- RunningSince
- NumberofEntries

`HTTP Method`: GET

`URI`: /status

`Content-Type`: application/json

6.9.1 Request

N/A

6.9.2 Response

A JSON document containing the aforementioned attributes

`Status Code` : OK / 200

Example Status Information

```
{
  "EMIRServerVersion":"1.2.2-SNAPSHOT",
  "MongoDBVersion":"2.0.4",
  "JavaVersion":"1.6.0_24",
  "OSName":"Linux",
  "OSArchitecture":"amd64",
  "OSVersion":"2.6.37.6-0.11-xen",
  "EMIRServerComponentName":"Domain Service Registry (DSR)",
```

```
  "AnonymousAccessPortNumber":"9127",
  "RunningSince":"Fri Jun 14 17:49:53 CEST 2013",
  "NumberofEntries":279
}
```

7 Appendix I

An example JSON document representing the Service endpoint records.

```
[       //Example Service Endpoints Records (belonging to the same service)
        {
                "Service_ID":"s1",
                "Service_Name":"ComputingService",
                "Service_CreationTime":{"$date":"2011-07-21T11:47:24Z"},
                "Service_Type":"job-management",
                "Service_Contact": [{"ContactType":"sysadmin", "Detail":"http://contactlink"},{"ContactType":"developer", "Detail":"http://contactlink"}],
                "Service_Endpoint_ID":"se1", //this should be unique
                "Service_Endpoint_URL":"http://1",
                "Service_Endpoint_Capability":["capability1","capability2"],
                "Service_Endpoint_Technology":"technology",
                "Service_Endpoint_InterfaceName":"interface",
                "Service_Endpoint_InterfaceVersion":["version1","version2"],
                "Service_Endpoint_InterfaceExtension":["extension1","extension2"],
                "Service_Endpoint_WSDL":"http//1.wsdl",
                "Service_Endpoint_SupportedProfile":["profile1","profile2"],
                "Service_Endpoint_Semantics":["semantic1","semantic2"],
                "Service_Endpoint_HealthState":"ok",
                "Service_Endpoint_HealthStateInfo":"state info",
                "Service_Endpoint_ServingState":"production",
                "Service_Endpoint_StartTime":{"$date":"2011-07-21T11:47:24Z"},
                "Service_Endpoint_DowntimeAnnounce":{"$date":"2011-07-21T11:47:24Z"},
                "Service_Endpoint_DowntimeStart":{"$date":"2011-07-21T11:47:24Z"},
                "Service_Endpoint_DowntimeEnd":{"$date":"2011-07-21T11:47:24Z"},
```

```
                "Service_Endpoint_QualityLevel":"production",
                "Service_Location_Address":"A Street 1",
                "Service_Location_Place":"Bonn",
                "Service_Location_Country":"Germany",
                "Service_Location_PostCode":"53119",
                "Service_Location_Latitude":53.3,
                "Service_Location_Longitude":4,
                "Service_ExpireOn":{"$date":"2020-07-21T11:47:24Z"}

        },
        {
                "Service_ID":"s1",
                "Service_Name":"ComputingService",
                "Service_CreationTime":{"$date":"2011-07-21T11 ↩
                    :47:24Z"},
                "Service_Type":"job-management",
                "Service_Contact": [{"ContactType":"sysadmin", " ↩
                    Detail":"http://contactlink"},{"ContactType":" ↩
                    developer", "Detail":"http://contactlink"}],
                "Service_Endpoint_ID":"se2", //this should be  ↩
                    unique
                "Service_Endpoint_URL":"http://1",
                "Service_Endpoint_Capability":["capability1"," ↩
                    capability2"],
                "Service_Endpoint_Technology":"technology",
                "Service_Endpoint_InterfaceName":"interface",
                "Service_Endpoint_InterfaceVersion":["version1"," ↩
                    version2"],
                "Service_Endpoint_InterfaceExtension":["extension1 ↩
                    ","extension2"],
                "Service_Endpoint_WSDL":"http//1.wsdl",
                "Service_Endpoint_SupportedProfile":["profile1"," ↩
                    profile2"],
                "Service_Endpoint_Semantics":["semantic1"," ↩
                    semantic2"],
                "Service_Endpoint_HealthState":"ok",
                "Service_Endpoint_HealthStateInfo":"state info",
                "Service_Endpoint_ServingState":"production",
                "Service_Endpoint_StartTime":{"$date":"2011-07-21 ↩
                    T11:47:24Z"},
                "Service_Endpoint_DowntimeAnnounce":{"$date ↩
                    ":"2011-07-21T11:47:24Z"},
                "Service_Endpoint_DowntimeStart":{"$date ↩
                    ":"2011-07-21T11:47:24Z"},
                "Service_Endpoint_DowntimeEnd":{"$date":"2011-07-21 ↩
                    T11:47:24Z"},
                "Service_Endpoint_QualityLevel":"production",
                "Service_Location_Address":"A Street 1",
                "Service_Location_Place":"Berlin",
                "Service_Location_Country":"Germany",
```

```
                "Service_Location_PostCode":"53011",
                "Service_Location_Latitude":53.5,
                "Service_Location_Longitude":4,
                "Service_ExpireOn":{"$date":"2020-07-21T11:47:24Z"}
        }

]
```

8 Appendix II

The normative form of the EMIR REST-ful API in Web Application Description Language (WADL)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<application xmlns="http://wadl.dev.java.net/2009/02">
        <doc xmlns:jersey="http://jersey.java.net/" jersey: ↩
            generatedBy="Jersey: 1.17 01/17/2013 03:31 PM" />
        <grammars />
        <resources base="https://localhost:54321/">
                <resource path="/status">
                        <method id="getServerStatus" name="GET">
                                <response>
                                        <representation mediaType=" ↩
                                            application/json" />
                                        <representation mediaType=" ↩
                                            text/plain" />
                                </response>
                        </method>
                </resource>
                <resource path="/parent">
                        <method id="childDSRs" name="GET">
                                <response>
                                        <representation mediaType ↩
                                            ="*/*" />
                                </response>
                        </method>
                </resource>
                <resource path="/ping">
                        <method id="ping" name="GET">
                                <response>
                                        <representation mediaType=" ↩
                                            application/json" />
                                        <representation mediaType=" ↩
                                            text/plain" />
                                </response>
                        </method>
                </resource>
```

```
                <resource path="/serviceadmin">
                        <method id="getServicebyID" name="GET">
                                <response>
                                        <representation mediaType=" ↩
                                            application/json" />
                                </response>
                        </method>
                        <method id="registerServices" name="POST">
                                <request>
                                        <representation mediaType=" ↩
                                            application/json" />
                                </request>
                                <response>
                                        <representation mediaType=" ↩
                                            application/json" />
                                </response>
                        </method>
                        <method id="updateServices" name="PUT">
                                <request>
                                        <representation mediaType=" ↩
                                            application/json" />
                                </request>
                                <response>
                                        <representation mediaType=" ↩
                                            application/json" />
                                </response>
                        </method>
                        <method id="deleteService" name="DELETE">
                                <response>
                                        <representation mediaType ↩
                                            ="*/*" />
                                </response>
                        </method>
                </resource>
                <resource path="/children">
                        <method id="childDSRs" name="GET">
                                <response>
                                        <representation mediaType ↩
                                            ="*/*" />
                                </response>
                        </method>
                        <method id="checkin" name="POST">
                                <response>
                                        <representation mediaType ↩
                                            ="*/*" />
                                </response>
                        </method>
                </resource>
                <resource path="/services">
                        <method id="queryWithParamsForJSON" name=" ↩
```

```
                    GET">
                        <response>
                                <representation mediaType=" ↩
                                    application/json" />
                        </response>
                    </method>
                    <method id="queryWithParamsForXML" name=" ↩
                        GET">
                        <response>
                                <representation mediaType=" ↩
                                    application/xml" />
                                <representation mediaType=" ↩
                                    text/xml" />
                        </response>
                    </method>
                    <method id="richQueryForJSON" name="POST">
                        <request>
                                <representation mediaType=" ↩
                                    application/json" />
                        </request>
                        <response>
                                <representation mediaType=" ↩
                                    application/json" />
                        </response>
                    </method>
                    <method id="richQueryForXML" name="POST">
                        <request>
                                <representation mediaType=" ↩
                                    application/json" />
                        </request>
                        <response>
                                <representation mediaType=" ↩
                                    application/xml" />
                        </response>
                    </method>
                    <resource path="/urls">
                        <method id="getServiceEndPoints"  ↩
                            name="GET">
                            <response>
                                    <representation  ↩
                                        mediaType=" ↩
                                        application/ ↩
                                        json" />
                            </response>
                        </method>
                    </resource>
                    <resource path="/types">
                        <method id="getServiceTypes" name=" ↩
                            GET">
                            <response>
```

```
                                        <representation  ↩
                                            mediaType=" ↩
                                            application/ ↩
                                            json" />
                                </response>
                        </method>
                </resource>
                <resource path="/query.xml">
                        <method id="queryXml" name="GET">
                                <response>
                                        <representation  ↩
                                            mediaType=" ↩
                                            application/xml ↩
                                            " />
                                        <representation  ↩
                                            mediaType="text ↩
                                            /xml" />
                                </response>
                        </method>
                </resource>
                <resource path="/pagedquery">
                        <method id="pagedQuery" name="GET">
                                <response>
                                        <representation  ↩
                                            mediaType=" ↩
                                            application/ ↩
                                            json" />
                                </response>
                        </method>
                        <method id="pagedQueryGlue2" name=" ↩
                            GET">
                                <response>
                                        <representation  ↩
                                            mediaType=" ↩
                                            application/xml ↩
                                            " />
                                </response>
                        </method>
                </resource>
        </resource>
        <resource path="/services/facet">
                <method id="getFacets" name="GET">
                        <response>
                                <representation mediaType=" ↩
                                    application/json" />
                        </response>
                </method>
        </resource>
        <resource path="/neighbors">
                <method id="childDSRs" name="GET">
```

```
                                <response>
                                        <representation mediaType ↩
                                            ="*/*" />
                                </response>
                        </method>
                </resource>
                <resource path="/model">
                        <method id="getModel" name="GET">
                                <response>
                                        <representation mediaType=" ↩
                                            text/html" />
                                        <representation mediaType=" ↩
                                            application/json" />
                                </response>
                        </method>
                </resource>
        </resources>
</application>
```